A Fault-Tolerant Exascale Parallel Runtime

Amos Waterland,[1,*] Jonathan Appavoo,[2] Elaine Angelino,[1] Ryan Adams,[1] and Margo Seltzer[1]
[1] *Harvard University*
[2] *Boston University*

Introduction

Major conceptual challenges still remain in making efficient use of highly parallel computing systems. Despite decades of sophisticated research, we still do not have computers that we can program in the sequential model best suited to the human mind, but whose performance automatically scales with the number of processors. As exascale computer systems appear on the horizon this becomes an increasingly critical challenge, and it seems clear that we will need new approaches.

The new approach we describe in this position paper is to transform sequential computation into a *statistical pattern recognition* and *caching* problem. We do this because pattern recognition and searching a distributed cache generally *does* scale with the number of processors and generally *is* fault-tolerant.

Based on our work at IBM Research on the Blue Gene/P team [1, 22], we believe that writing explicitly parallel fault-tolerant programs for exascale systems is unlikely to work for any but a small class of applications. We propose that exascale systems *can* be sequentially programmed in a fault-oblivious manner *if* large quantities of information about the runtime structure of sequential programs can be reasoned about and exploited. Manually formalizing all this information through traditional approaches, which rely on semantic analysis at the language level, has historically proved challenging. We are therefore taking a lower level approach, delaying explicit semantic analysis and instead first modeling von Neumann computation as a *dynamical system*, i.e., a state space and an evolution rule, which gives a natural way to use statistical inference to then automatically *learn* powerful representations of this information that carry their own inferred semantics. This is in rough analog to the painful lesson learned by the computer vision community, in which heroic but mostly failed work was done to impose human-level semantics on raw images in order to recognize objects. The statistical pattern recognition approach arose—and now dominates vision research—by realizing that much more powerful image features can be *learned* than can be imposed by human experts.

Our model gives us a promising new approach to making efficient use of highly parallel computing systems – we use probability distributions and symmetry transforms *empirically learned* over the state space and a massively parallel pool of machine simulators to achieve automatic speedup though a speculative, generalized form of memoization. We have built a prototype MPI program—currently in operation on the Intrepid 40-rack Blue Gene/P system at Argonne National Lab—that uses this model of computation to automatically achieve linear speedups for some simple sequential binary programs on up to 32,768 nodes.

In our model, the complete state of a computer is represented as the coordinates of a point in our *state space*. Computation is effected by repeatedly applying our *evolution rule*, which maps each point in the state space to its successor point by simulating the instruction encoded in the coordinates. A sequence of such transitions forms a path through state space called a *trajectory*. Some trajectories terminate in *fixed points* – points that are mapped to themselves by the evolution rule. The result of computation is obtained by waiting until the trajectory being solved halts at its fixed point, then reading out the answer from the coordinates. Programming is effected by preparing an *initial condition* – selecting a point in state space whose coordinates represent the initial values of the registers, machine code, and data. The state of computation is just a *vector*, a trace of computation is just a *matrix*, and we have a *geometry* of computation complete with *distance* and *angle* between states of computation. This model gives a natural way to apply *inference*, in the form of Bayesian posterior maximization and deep artificial neural networks, through highly parallel execution of Markov Chain Monte Carlo, simulated annealing, and genetic algorithms.

In our prototype MPI program, each node implements a machine simulator that has the same state space and evolution rule, but is given different initial conditions. Our prototype is invoked as *e.g.* `mpirun -np 32768 runtime program`, where `runtime` is our prototype and `program` is a sequential binary that we want to accelerate. The nodes behave as a single giant runtime that *cooperates* in parallel to accelerate the execution of the supplied program. Each node maintains a state vector representation of its simulated computer, and has a simulation loop that calls the evolution rule – which simulates one instruction per invocation. Each node also maintains a compressed cache of *initial* and *final points* of state space trajectories it has already solved, and when it has no other work to do, garbage collects and updates its cache by speculatively solving trajectories given *predictions* for regions of state space that other

nodes are likely to visit.

At regular intervals, each node *broadcasts* its current state vector to the other nodes, all of whom in response search their caches for a match. A match is found when the broadcast state vector is equivalent under a *symmetry transform* to the initial point of a previously solved trajectory. This trajectory was either speculatively solved given a prediction or lies on an unrelated trajectory whose initial condition is a different program that for a time executed a sequence of instructions identical to those of the current program. In both cases, the node that found a match applies the inverse symmetry transform and replies with the final point of the matched trajectory.

Upon receiving the reply, the node that sent the broadcast then jumps forward—in state space and simulation time—from its current point to the final point it received in reply. This instantaneous jump through state space—which skips over the many applications of the evolution rule that it would have had to do—is called *tunneling*. There is no free lunch, as some node somewhere had to solve each trajectory up to symmetries, but from the perspective of the node that sent the broadcast – it has been accelerated.

Tunneling can be seen as a speculative, generalized form of *memoization*. Traditional memoization speeds up programs by building a table of the inputs and outputs of *pure functions*. Once a memoized function has been executed for a particular input, it never has to be executed again when called with the same input; the result can simply be looked up in the table. Tunneling is a generalization of memoization in that it can jump forward from *any* program location – not just at a function boundary, and it can use results from one program to speed up a different program. It is speculative in that instead of just storing the results of computation that has already happened, it solves trajectories in the hope that they will be useful later to other nodes. These tunnels can be thought of as warping the state space of computation so that useful and important trajectories are drastically compressed.

This system design is probabilistic, in that it uses Bayesian inference to calculate the predictive probability distributions used by speculative trajectory solvers, but it is not probabilistic or approximate computing in the sense that it might halt with the wrong result. When predictions are poor, the worst that can happen is that communication and simulation overhead is not recovered. Error-correcting codes are used to ensure that the system ignores incorrect cache hits returned by failing nodes, and the robust nature of statistical pattern recognition means that node faults can be "trained around". We preserve the deterministic sequential programming model, our initial conditions are prepared by gcc, and the result of simulation is identical to that of running the input binary program on a uniprocessor computer. Our use of Bayesian inference also gives a natural division of work on heterogeneous parallel computers – in which speculative trajectory solvers run on full-featured cores, but feature extraction and predictive inference is done by massive arrays of simple processors in GPUs or neuromorphic devices.

Related work

A number of physicists have previously observed that computers can be abstractly modeled as dynamical systems [2, 6, 7, 9, 14, 24]. However, to our knowledge we are the first to show how to *operationalize* this theory into a concrete practical application that in principle allows sequential computation performance to scale with node count. The idea of using runtime statistics to accelerate programs is of course an active area of research [3, 4, 12, 18]; our work can be seen as somewhat related to Thread-Level Speculation [5, 17, 21], Decoupled Software Pipelining [16, 23, 25], memoizing processors [11, 13, 15, 20], advanced virtual machines [19, 26], caching cellular automata [8], and perceptron branch prediction [10].

Challenges addressed

Explicit fault-tolerant parallel programming of exascale systems will be extremely challenging. Our approach attempts to do an end-run around this challenge through the use of statistical pattern recognition and distributed cache search – which are known to generally scale with the number of nodes and to be fault-tolerant.

Maturity

This is early work, but we recently published a paper on our current results[27] on up to 32,768 cores.

Uniqueness

This approach is somewhat unique to exascale systems because it is there that the challenge of explicit parallel programming is most acute. In addition, deep neural networks and distributed caches benefit greatly from exascale systems.

Novelty

Scientific progress is sometimes made by turning hard problems into different hard problems for which we have better mathematical tools. We have turned the problem of accelerating sequential von Neumann computation into one of predictive inference in a dynamical system. Bayesian predictive inference has seen rapid growth in recent decades due to increased high-performance computing resources, but turning inference on von Neumann computation *itself* into an exascale high-performance computing job has received very little attention.

Applicability

Our model of computation has received interest from programming language professors interested in finding bugs via trajectory divergence.

Effort

We have three graduate students working on this project full-time; advised by three professors.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Fellow ID 2012116808 and the Department of Energy Office of Science under its agreement number DE-SC0005365.

* Electronic address: apw@seas.harvard.edu

[1] Jonathan Appavoo, Volkmar Uhlig, and Amos Waterland, *Project kittyhawk: building a global-scale computer: Blue gene/p as a generic computing platform*, Operating Systems Review **42** (2008), no. 1, 77–84.

[2] Henry G. Baker, *Thermodynamics and garbage collection*, SIGPLAN Not. **29** (1994), no. 4, 58–63.

[3] Vasanth Bala, Evelyn Duesterwald, and Sanjeev Banerjia, *Dynamo: a transparent dynamic optimization system*, ACM SIGPLAN Notices **35** (2000), no. 5, 1–12.

[4] Sorav Bansal and Alex Aiken, *Automatic generation of peephole superoptimizers*, SIGPLAN Not. **41** (2006), no. 11, 394–403.

[5] Anasua Bhowmik and Manoj Franklin, *A general compiler framework for speculative multithreading*, Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures (New York, NY, USA), SPAA '02, ACM, 2002, pp. 99–108.

[6] Roger W. Brockett, *Dynamical systems that sort lists, diagonalize matrices and solve linear programming problems*, Proc. 27th IEEE Conf. Dec. and Control (Austin, TX), Dec. 1988, pp. 799–803.

[7] Marco Giunti, *Computation, dynamics, and cognition*, Oxford University Press, 1997.

[8] Bill Gosper, *Exploiting regularities in large cellular spaces*, Physica D. Nonlinear Phenomena (1984).

[9] John J. Hopfield, *Hopfield network*, Scholarpedia **2** (2007), no. 5.

[10] Daniel A. Jimenez and Calvin Lin, *Dynamic branch prediction with perceptrons*, 2001.

[11] Y. Kamiya, T. Tsumura, H. Matsuo, and Y. Nakashima, *A speculative technique for auto-memoization processor with multithreading*, Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on, dec. 2009, pp. 160 –166.

[12] Henry Massalin, *Synthesis: an efficient implementation of fundamental operating system services*, Ph.D. thesis, New York, NY, USA, 1992, UMI Order No. GAX92-32050.

[13] Donald Michie, *Memo functions and machine learning*, Nature.

[14] Todd Mytkowicz, Amer Diwan, and Elizabeth Bradley, *Computer systems are dynamical systems*, Chaos: An Interdisciplinary Journal of Nonlinear Science **19** (2009), no. 3, 033124.

[15] Zach Purser, Karthik Sundaramoorthy, and Eric Rotenberg, *A study of slipstream processors*, Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture (New York, NY, USA), MICRO 33, ACM, 2000, pp. 269–280.

[16] Arun Raman, Hanjun Kim, Thomas R. Mason, Thomas B. Jablin, and David I. August, *Speculative parallelization using software multi-threaded transactions*, Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems (New York, NY, USA), ASPLOS '10, ACM, 2010, pp. 65–76.

[17] Lawrence Rauchwerger and David Padua, *The lrpd test: speculative run-time parallelization of loops with privatization and reduction parallelization*, Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation (New York, NY, USA), PLDI '95, ACM, 1995, pp. 218–232.

[18] Eric Rotenberg, Steve Bennett, and James E. Smith, *Trace cache: A low latency approach to high bandwidth instruction fetching*, International Symposium on Microarchitecture, 1996, pp. 24–35.

[19] Emin Gün Sirer, Robert Grimm, Arthur J. Gregory, and Brian N. Bershad, *Design and implementation of a distributed virtual machine for networked computers*, SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles (New York, NY, USA), ACM, 1999, pp. 202–216.

[20] Gurindar S. Sohi, Scott E. Breach, and T. N. Vijaykumar, *Multiscalar processors*, Proceedings of the 22nd annual international symposium on Computer architecture (New York, NY, USA), ISCA '95, ACM, 1995, pp. 414–425.

[21] J. Gregory Steffan, Christopher Colohan, Antonia Zhai, and Todd C. Mowry, *The stampede approach to thread-level speculation*, ACM Trans. Comput. Syst. **23** (2005), no. 3, 253–300.

[22] Jan Stoess, Udo Steinberg, Volkmar Uhlig, Jens Kehne, Jonathan Appavoo, and Amos Waterland, *A lightweight virtual machine monitor for blue gene/p*, IJHPCA **26** (2012), no. 2, 95–109.

[23] William Thies, Vikram Chandrasekhar, and Saman Amarasinghe, *A practical approach to exploiting coarse-grained pipeline parallelism in c programs*, Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (Washington, DC, USA), MICRO 40, IEEE Computer Society, 2007, pp. 356–369.

[24] Tommaso Toffoli, *Action, or the fungibility of computation*, pp. 349–392, Perseus Books, Cambridge, MA, USA, 1999.

[25] Neil Vachharajani, Ram Rangan, Easwaran Raman, Matthew J. Bridges, Guilherme Ottoni, and David I. August, *Speculative decoupled software pipelining*, Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (Washington, DC, USA), PACT '07, IEEE Computer Society, 2007, pp. 49–59.

[26] Chad Verbowski, Emre Kiciman, Arunvijay Kumar, Brad Daniels, Shan Lu, Juhan Lee, Yi-Min Wang, and Roussi Roussev, *Flight data recorder: Monitoring persistent-state interactions to improve systems management.*, OSDI, USENIX Association, 2006, pp. 117–130.

[27] Amos Waterland, Jonathan Appavoo, and Margo Seltzer, *Parallelization by simulated tunneling*, Proceedings of the USENIX Workshop on Hot Topics in Parallelism (HotPAR) (2012).